intel®

APPLICATION NOTE

AP-51

March 1979

Designing 8086, 8088, 8089 Multiprocessing Systems with the 8289 Bus Arbiter

Designing 8086, 8088, 8089 Multiprocessor Systems with the 8289 Bus Arbiter

Contents

Our thanks to Jim Nadir, the author of this application note. Jim is a design engineer in the microprocessors and peripherals operation division. Please direct any technical questions you may have to your local Intel FAE (Field Application Engineer).

INTRODUCTION

Over the past several years, microprocessors have been increasing in popularity. The performance improvements and cost reductions afforded by LSI technology have spurred on the design motivation of using multiple processors to meet system real-time performance requirements. The desire for improved system real-time response, system reliability and modularity has made multiprocessing techniques an increasingly attractive alternative to the system design engineer; techniques that are characterized as having more than one microprocessor share common resources, such as memory and I/O, over a common multiple processor bus.

This type of design concept allows the system designer to partition overall system functions into tasks that each of several processors can handle individually to increase system performance and throughput. But, how should a designer proceed to implement a multiprocessing system? Should he design his own? If so, how are the microprocessors synchronized to avoid contention problems? The designer could put them all in phase using one clock for all the microprocessors. This may work, until the physical dimensions of the system become large. When this occurs, the designer is faced with many problems, like clock skew (resulting in bus spec violations) and duty cycle variations.

A better approach to implementing a multiprocessor system is not to have a common processor clock, but allow each processor to work asynchronously with respect to each other. The microprocessor requests to use the multiple processor bus could then be synchronized to a high frequency external clock which will permit duty cycle and phase shift variations. This type of approach has the benefit of allowing modularity of hardware. When new system functions are desired, more processing power can be added without impacting existing processor task partitioning.

One approach to implement this asynchronous processing structure would be to have all the bus requests enter a priority encoder which samples its inputs as a function of the higher frequency "bus clock". The inputs would arrive asynchronously to the priority encoder and would be resolved by the priority encoder structure as to which microprocessor would be granted the bus. Another approach, that used by Intel, is rather than allowing the requests to arrive asynchronously with respect to one another at the priority encoder, the bus requests are synchronized first to an external high frequency bus clock and then sent to the priority encoder to be resolved. In this way, the resolving circuitry common to all microprocessors is kept at a minimum. Overall system reliability is improved in the sense that should a circuit which serves to synchronize the processor's request (which is now located on the same card as the microprocessor itself) fail, it is only necessary to remove that card from the system and the rest of the system will continue to function. Whereas in the other approach, should the synchronizing mechanism fail, the whole system goes down, as the synchronizing mechanism is located at the shared resource. In addition to the improved system reliability, moving the synchronization mechanism to the processor permits processor control over that mechanism, thereby permitting system flexibility (as will be shown) which could not be reasonably obtained by any other approach.

This synchronizing or arbitrating function was integrated into the 8289, a custom arbitration unit for the 8086, 8088, and 8089 processors. This note basically describes the 8289 arbitration unit, illustrates its different modes of operation and hardware connect in a multiprocessor system. Related and useful documents are: 8086 user's manual, 8289 data sheet, Article Reprint -55: Design Motivations for Multiple Processor Microcomputer Systems (which discusses implementing a semaphore with the MULTIBUS™) and Application Note 28A, Intel MULTIBUS™ interfacing.

BUS ARBITER OPERATING CHARACTERISTICS

The 8289 Bus Arbiter operates in conjunction with the 8288 Bus Controller to interface an 8086, 8088, or 8089 processor to a multi-master system bus (the 8289 is used as a general bus arbitration unit). The processor is unaware of the arbiter's existence and issues commands as though it has exclusive use of the system bus. If the processor does not have the use of the multimaster system bus, the bus arbiter prevents the bus controller, the data transceivers and the address latches from accessing the system bus (i.e., all bus driver outputs are forced into the high impedance state). Since the command was not issued, a transfer acknowledge (XACK) will not be returned and the processor will enter into wait states. Transfer acknowledges are signals returned from the addressed resource to indicate to the processor that the transfer is complete. This signal is typically used to control the ready inputs of the clock generator. The processor will remain in wait until the bus arbiter acquires the use of the multi-master system bus, whereupon the bus arbiter will allow the bus controller, the data transceivers and the address latches to access the system bus. Once the command has been issued and a data transfer has taken place, a transfer acknowledge (XACK) is returned to the processor. The processor then completes its transfer cycle. Thus, the arbiter serves to multiplex a processor (or bus master) onto a multi-master system bus and avoid contention problems between bus masters.

Since there can be many bus masters on a multi-master system bus, some means of resolving priority between bus masters simultaneously requesting the bus must be provided. The 8289 Bus Arbiter provides for several resolving techniques. All the techniques are based on a priority concept that at a given time one bus master will have priority above all the rest. These techniques include the parallel priority resolving techniques, serial priority resolving and rotating priority techniques.

A parallel priority resolving technique has a separate bus request ($\overline{BREQ}$) line for each arbiter on the multi-master bus (see Figure 1). Each $\overline{BREQ}$ line enters into a priority encoder which generates the binary address of the highest priority $\overline{BREQ}$ line which is active at the inputs. The output binary address is decoded by a decoder to select the corresponding $\overline{BPRN}$ (bus priority in) line to be returned to the highest priority requesting arbiter. The arbiter receiving priority ($\overline{BPRN}$ active low) then allows its associated bus master onto the multi-master system bus as soon as it becomes available (i.e., it is no longer busy). When one bus arbiter gains priority over another arbiter, it cannot immediately seize the bus, it must wait until the present bus occupant completes its transfer cycle. Upon completing its transfer cycle, the present bus occupant recognizes that it no longer has priority and surrenders the bus, releasing $\overline{BUSY}$. $\overline{BUSY}$ is an active low OR-tied signal line which goes to every bus arbiter on the system bus. When $\overline{BUSY}$ goes high, the arbiter which presently has bus priority ($\overline{BPRN}$ active low) then seizes the bus and pulls $\overline{BUSY}$ low to keep other arbiters off the bus. (See waveform timing diagram, Figure 2.) Note that all multi-master system bus transactions are synchronized to the bus clock ($\overline{BCLK}$). This allows for the parallel priority resolving circuitry or, any other priority resolving scheme employed, time to settle and make a correct decision.

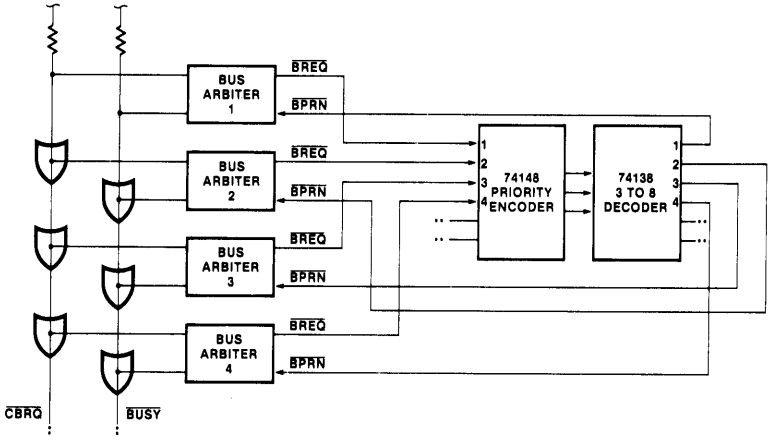
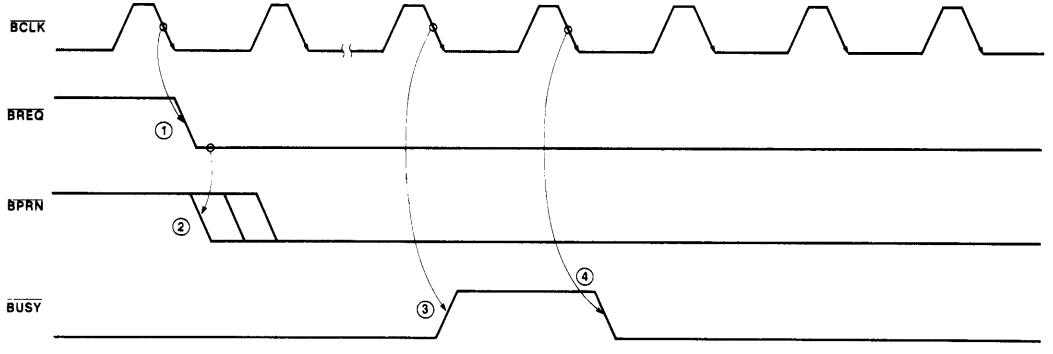


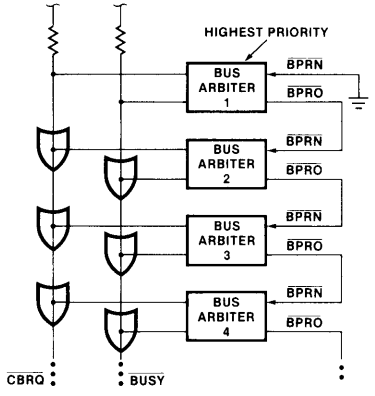
Figure 1. Parallel Priority Resolving Technique



1. HIGHER PRIORITY BUS ARBITER REQUESTS THE MULTI-MASTER SYSTEM BUS.
2. ATTAINS PRIORITY.
3. LOWER PRIORITY BUS ARBITER RELEASES BUSY.
4. HIGHER PRIORITY BUS ARBITER THEN ACQUIRES THE BUS AND PULLS BUSY DOWN.

Figure 2. Higher Priority Arbiter Obtaining The Bus From A Lower Priority Arbiter

A serial priority resolving technique eliminates the need for the priority encoder-decoder arrangement by daisy-chaining the bus arbiters together. This is accomplished by connecting the higher priority bus arbiter's $\overline{BPRO}$ (bus priority out) output to the $\overline{BPRN}$ of the next lower priority (see Figure 3). The highest priority bus arbiter would have its $\overline{BPRN}$ line grounded, signifying to the arbiter that it always has highest priority when requesting the bus.



THE NUMBER OF ARBITERS THAT MAY BE DAISY-CHAINED TOGETHER IN THE SERIAL PRIORITY RESOLVING TECHNIQUE IS A FUNCTION OF $\overline{BCLK}$ AND THE PROPAGATION DELAY FROM ARBITER TO ARBITER. NORMALLY, AT 10 MHz ONLY 3 ARBITERS MAY BE DAISY-CHAINED. SEE TEXT.

Figure 3. Serial Priority Resolving

A rotating priority resolving technique arrangement is similar to that of the parallel priority resolving technique except that priority is dynamically reassigned. The priority encoder is replaced by a more complex circuit which rotates priority between requesting arbiters, thus guaranteeing each arbiter equal time on the multi-master system bus.

There are advantages and disadvantages for each of the techniques described above. The rotating priority resolving technique requires an extensive amount of logic to implement, while the serial technique can accommodate only a limited number of bus arbiters before the daisy-chain propagation delay exceeds the multi-master system bus clock ($\overline{BCLK}$). The parallel priority resolving technique is, in general, the best compromise. It allows for many arbiters to be present on the bus while not requiring much logic to implement.

Whatever resolving technique is chosen, it is the highest priority bus arbiter requesting use of the multi-master system bus which obtains the bus. Exceptions do exist with the 8289 Bus Arbiter where a lower priority arbiter may take away the bus from a higher priority arbiter without the need for any additional external logic. This is accomplished through the use of the $\overline{CBRQ}$ pin, discussed in a later section.

MULTI-MASTER SYSTEM BUS SURRENDER AND REQUEST

The 8289 Bus Arbiter provides an intelligent interface to allow a processor or bus master of the 8086 family to access a multi-master system bus. The arbiter directs the processor onto the bus and allows both higher and lower priority bus masters to acquire the bus. Higher priority masters obtain the bus when the present bus master utilizing the bus completes its transfer cycle (including hold time). Lower priority bus masters obtain the bus when a higher priority bus master is not accessing the system bus and a lower priority arbiter has pulled $\overline{CBRQ}$ low. This signifies to the arbiter presently holding the multi-processor bus that a lower priority arbiter would like to acquire the bus when it is not being used. A strapping option (ANYRQST) allows the multi-master system bus to be surrendered to any bus master requesting the bus, regardless of its priority. If there are no other bus masters requesting the bus, the arbiter maintains the bus as long as its associated bus master has not entered the HALT state. *The 8289 Bus Arbiter will not voluntarily surrender the system bus and has to be forced off by another bus master.* An exception to this can be obtained by strapping $\overline{CBRQ}$ low and ANYRQST high. In this configuration the 8289 will release the bus after each transfer cycle.

How the 8289 Bus Arbiter is configured determines the manner in which the arbiter requests and surrenders the system bus. If the arbiter is configured to operate with a processor which has access to both a multi-master system bus and a resident bus, the arbiter requests the use of the multi-master system bus only for system bus accesses (i.e., it is a function of the SYSB/$\overline{RESB}$ input pin). While the processor is accessing the resident bus, the arbiter permits a lower priority bus master to seize the system bus via $\overline{CBRQ}$, since it is not being used. A processor configuration with both an I/O peripheral bus and a system bus behaves similarly. If the processor is accessing the peripheral bus, the arbiter permits the surrendering of the multi-master system bus to a lower priority bus master. To request the use of the multi-master system bus, the processor must perform a system memory access (as opposed to an I/O access).

The arbiter decodes the processor status lines to determine what type of access is being performed and behaves correspondingly. For simpler system configurations, such as a processor which accesses only a multi-master system bus, the arbiter requests the use of the system bus when it detects the status lines initiating a transfer cycle. The decoding of these status lines can be referenced in the 8086, 8088 (non-I/O processor) data sheets or the 8089 (I/O processor) data sheet.

There is one condition common to all system configurations where the multi-master system bus is surrendered to a lower priority bus master requesting the bus by pulling $\overline{CBRQ}$ low. This is the idle or inactive state (TI) which is unique to the 8086 and 8088 processor family. This TI state comes about due to the processor's ability to fetch instructions in advance and store them internally for quick access. The size of the internal queue was optimized so that the processor would make the most ef-

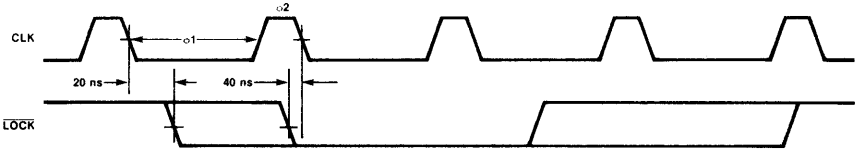
fective use of its resources and be slightly execution bound. Since the processor can fetch code faster than it can execute it, it will fill to capacity its internal storage queue. When this occurs, the processor will enter into idle or inactive states (TI) until the processor has executed some of the code in the storage queue. Once this occurs, the processor will exit the TI state and again start code fetching. Between entering into and exiting from the TI state an indeterminate number of TI states can occur during which the bus arbiter permits the surrendering of the multi-master system bus to a lower priority bus master. As noted earlier and worth repeating here, once the 8289 Bus Arbiter acquires the use of the multi-master system it will not voluntarily surrender the bus and has to be forced off by another bus master. This will be discussed in more detail later.

Two other signals, $\overline{\text{LOCK}}$ and $\overline{\text{CRQLCK}}$ (Figure 4), lend to the flexibility of the 8289 Bus Arbiter within system configurations. $\overline{\text{LOCK}}$ is a signal generated by the processor to prevent the bus arbiter from surrendering the multi-master system bus to any other bus master, either higher or lower priority. $\overline{\text{CRQLCK}}$ (common request lock) serves to prevent the bus arbiter from surrendering the bus to a lower priority bus master when conditions warrant it. $\overline{\text{LOCK}}$ is used for implementing software semaphores for critical code sections and real time critical events (such as refreshing or hard disk transfers).

8289 BUS ARBITER INTERFACING TO THE 8288 BUS CONTROLLER

Once the 8289 Bus Arbiter determines to either allow its associated processor onto the multi-master system bus or to surrender the bus, it must guarantee that command setup and hold times are not violated. This is a two part problem. One, guaranteeing hold time and two, guaranteeing setup time. The 8288 Bus Controller performs the actual task of establishing setup time, while the 8289 Bus Arbiter establishes hold time (see Figure 5).

The 8289 Bus Arbiter communicates with the 8288 Bus Controller via the $\overline{\text{AEN}}$ line. When the arbiter allows its associated processor access to the multi-master system bus, it activates $\overline{\text{AEN}}$. $\overline{\text{AEN}}$ immediately enables the address latches and data transceivers. The bus controller responds to $\overline{\text{AEN}}$ by bringing its command output buffers out of high impedance state but keeping all commands disqualified until command setup time is established. Once established, the appropriate command is then issued. $\overline{\text{AEN}}$ is brought to the false state after the command hold time has been established by the arbiter when surrendering the bus.

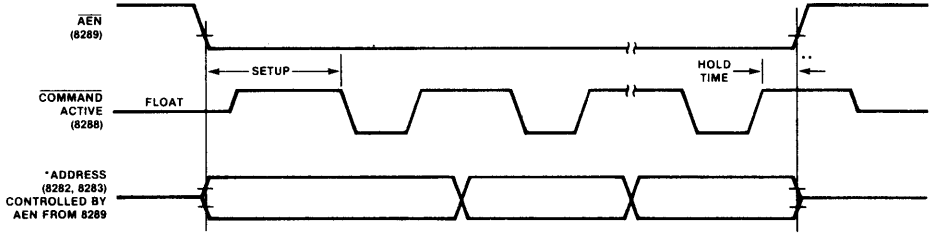


LOCK TIMING

THE ONLY CRITICAL LOCK TIMING IS THAT SHOWN ABOVE. $\overline{\text{LOCK}}$ MUST BE ACTIVATED NO SOONER THAN 20 ns INTO φ1 AND NO LATER THAN 40 ns PRIOR TO THE END OF φ2. $\overline{\text{LOCK}}$ INACTIVE HAS NO CRITICAL TIMING AND CAN BE ASYNCHRONOUS.

$\overline{\text{CRQLCK}}$ HAS NO CRITICAL TIMING AND IS CONSIDERED AS AN ASYNCHRONOUS INPUT SIGNAL.

Figure 4. Lock Timing



*ADDRESSES ARE ACTIVATED IMMEDIATELY WHILE COMMAND IS DELAY TO ESTABLISH SETUP TIME REQUIREMENTS.

**THE 8289 ARBITER INTERNALLY TRACKS THE PROCESSOR CYCLE TO ESTABLISH THE PROPER AMOUNT OF HOLD TIME AFTER THE COMMAND HAS GONE INACTIVE.

Figure 5. Single Bus Interface Timing

8289 BUS ARBITER INTERNAL ARCHITECTURE

A block diagram of the internal architecture of the 8289 Bus Arbiter is shown in Figure 6. It is useful to understand this block diagram when discussing the different modes of the 8289 and their impact on processor bus operations; however, you may want to skip this section to "8086 family processor types and system configurations" and return to it afterwards, as this section addresses the very involved reader. The front end state generator (FETG) and the back end state generator (BETG) allow the arbiter to track the processor cycle. An examination of an 8086 family processor state timings show that all command and control signals are issued in states T1 and T2 while being terminated in states T3 and T4, with an indeterminate number of wait states (Tw) occurring in between. Note further, that an indeterminate number of idle or inactive states can occur immediately proceeding and following a given transfer cycle. Since an indeterminate number of wait states can occur, two state generators are required; one to generate control signals (the FETG) and one to terminate control signals (the BETG). The FETG is triggered into operation when the processor activates the status lines. The FETG is reset and the BETG is triggered into operation by the status lines going to the passive condition. The BETG is reset when the status lines again go active.

It is necessary for the 8289 Bus Arbiter to track the processor in order that it is properly able to determine where and when to request or surrender the use of the multi-master system bus. In system configurations which access a resident bus, the use of the multi-master system bus is requested later in order to allow time for the SYSB/$\overline{\text{RESB}}$ input to become valid. For systems which access a peripheral bus, the arbiter issues a request for the system bus only for memory transfer cycles which it decodes from the status lines (and time must be allowed for the status lines to become valid and then decoded). In a system which accesses only a multi-master system bus, a request is made as soon as the arbiter detects an active-going transition on the processor's status lines. Thus, when the processor initiates a transfer cycle, the FETG is triggered into operation and, depending upon what mode the arbiter is configured in, the STATUS & MODE DECODE circuitry initiates a request for the system bus at the appropriate time. The request enters the BREQ SET circuitry where it is then synchronized to the multi-master system bus clock ($\overline{\text{BCLK}}$) by the PROCESSOR SYNCHRONIZATION circuitry.* Once synchronized, the multi-master system bus interface circuitry issues a $\overline{\text{BREQ}}$. When the priority resolving circuitry returns a $\overline{\text{BPRN}}$ (bus priority in), the PROCESSOR SYNCHRONIZATION circuitry seizes the bus the next time it becomes available (i.e., $\overline{\text{BUSY}}$ goes high) by pulling $\overline{\text{BUSY}}$ low one $\overline{\text{BCLK}}$ after it goes high and enables $\overline{\text{AEN}}$. (See waveform timing diagram in Figure 2). Once the arbiter acquires the use of the system bus and a data exchange has taken place (a transfer acknowledge, XACK, was returned to the processor), the processor status lines go passive and the

*Due to the asynchronous nature of processor trasnsfer request to the multi-master system bus clock, it is necessary to synchronize the processor's transfer request to BCLK.

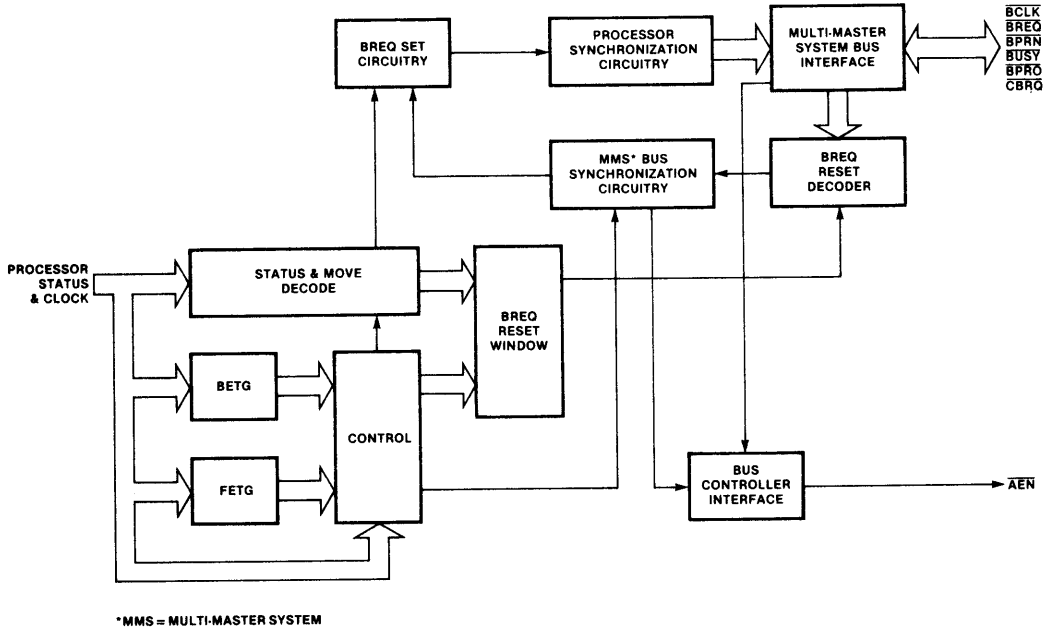


Figure 6. 8289 Bus Arbiter Block Diagram

BETG is triggered into operation. The BETG provides the timing for the bus surrender circuitries in the event that conditions warrant the surrender of the multi-master bus, i.e., the bus arbiter lost priority to a higher bus master or the processor has entered into TI states and $\overline{CBRQ}$ is pulled low, etc. If such is the case, the BREQ RESET DECODER initiates a bus surrender request. The bus surrender request is synchronized by the MMS BUS SYNCHRONIZATION CIRCUITRY to the processor clock. The MMS BUS SYNCHRONIZATION CIRCUITRY instructs the bus controller interface circuitry to make $\overline{AEN}$ go false and resets the BREQ SET circuitry. Resetting the BREQ SET circuitry will cause its output to go false and be synchronized by the processor synchronization, eventually instructing the MULTI-MASTER SYSTEM BUS INTERFACE circuitry to reset $\overline{BREQ}$. In the event that a lower priority arbiter has caused the arbiter to surrender the bus, it is necessary that $\overline{BREQ}$ be reset. Resetting $\overline{BREQ}$ allows the priority resolving circuitry to generate $\overline{BPRN}$ to the next highest priority bus master requesting the bus. The BREQ RESET WINDOW circuitry provides a 'window' wherein the arbiter allows the multi-master system bus to be surrendered and serves as part of the MMS bus-processor synchronization circuitry.

8086 FAMILY PROCESSOR TYPES AND SYSTEM CONFIGURATIONS

There are two types of processors in the 8086 family — an I/O processor (the 8089 IOP) and a non-I/O processor (the 8086 and 8088 CPUs). Consequently, there are two basic operating modes in the 8289 Bus Arbiter. One, the IOB (I/O peripheral bus) mode, permits the processor access to both an I/O peripheral bus and a multi-master system bus. The second, the RESB (resident bus) mode, permits the processor to communicate over both a resident bus and a multi-master system bus. Even though it is intended for the arbiter to be configured in the IOB mode when interfacing to an I/O processor and for it to be in the RESB mode when interfacing to a non-I/O processor, it is quite possible for the reverse to be true. That is, it is possible for a non-I/O processor to have access to an I/O peripheral bus or for an I/O processor to have access to a resident bus as well as access to a multi-master system bus. The IOB strapping option configures the 8289 Bus Arbiter into the IOB mode and RESB strapping option configures it into the resident bus mode. If both strapping options are strapped false, a third mode of operation is created, the single bus mode, in which the arbiter interfaces the processor to a multi-master system bus only. With both options strapped true, the arbiter interfaces the processor to a multi-master system bus, a resident bus and an I/O bus.

To better understand the 8289 Bus Arbiter, each of the operating modes, along with their respective timings, are examined by means of examples. The simplest configuration, the Single Bus Configuration, (both IOB and RESB strapped inactive) will be considered first, followed by the I/O bus Configuration and the Resident Bus Configuration. Finally, brief mention is made of a configuration that allows the processor to interface to two multi-master system buses. This particular configuration is briefly mentioned because, as will be seen, it is simply an extension of the resident bus configuration. When discussing the Single Bus Configuration, processor/arbiter, arbiter/system bus and internal arbiter, considerations are made resulting in a table that illustrates overhead in requesting the system bus. As this applies to the other 8289 configurations, only additional considerations will be given. A summary of when to use the different configurations is given at the end.

8289 SINGLE BUS INTERFACE

Figure 7 shows a block diagram of a bus master which has to interface only to a system bus — preferably the MULTIBUS — where there exists more than one bus master. In later configurations, it will be shown how the processor can be made to interface with more than one bus. Since the processor has only to interface with one bus, this configuration is called "single".

Connecting the 8289 Bus Arbiter to the processor is as simple as it was to connect the 8288 Bus Controller. Namely, the three status lines, $\overline{S0}$, $\overline{S1}$, and $\overline{S2}$ are directly connected from the processor to the arbiter. The clock line from the 8284 Clock Generator is brought down and connected. (Note that both the 8288 Bus Controller and the 8289 Bus Arbiter are connected to the same clock, CLK and not the peripheral clock, PCLK as the 8086 processor.) From the arbiter, $\overline{AEN}$ is connected to the bus controller and to the clock generator. The $\overline{IOB}$ pin on the arbiter is strapped high and on the controller the IOB pin is strapped low. In addition, the RESB pin on the arbiter is strapped low, finishing the processor interface.

Some flexibility exists with the MULTIBUS or multi-master system bus interface. The system designer must first decide upon the type of priority resolving scheme to be employed, whether it is to be the serial, parallel, or rotating priority scheme. A rotating priority scheme would be employed where the system designer would want to guarantee that every bus master on the bus would be given time on the bus. In the serial and parallel schemes, the possibility exists that the lowest assigned priority bus master may not acquire the bus for long periods of time. This occurs because priority is permanently assigned and if bus demand is high by the higher assigned priorities, then the lower priorities must wait. In most cases, this situation is acceptable because the highest priority is assigned to the bus master that cannot wait. Highest priority is usually assigned to DMA type devices where service requirements occur in real time. CPUs are assigned the lower priorities. For the purpose of this discussion, the parallel priority scheme will be used with brief reference to the serial priority scheme.

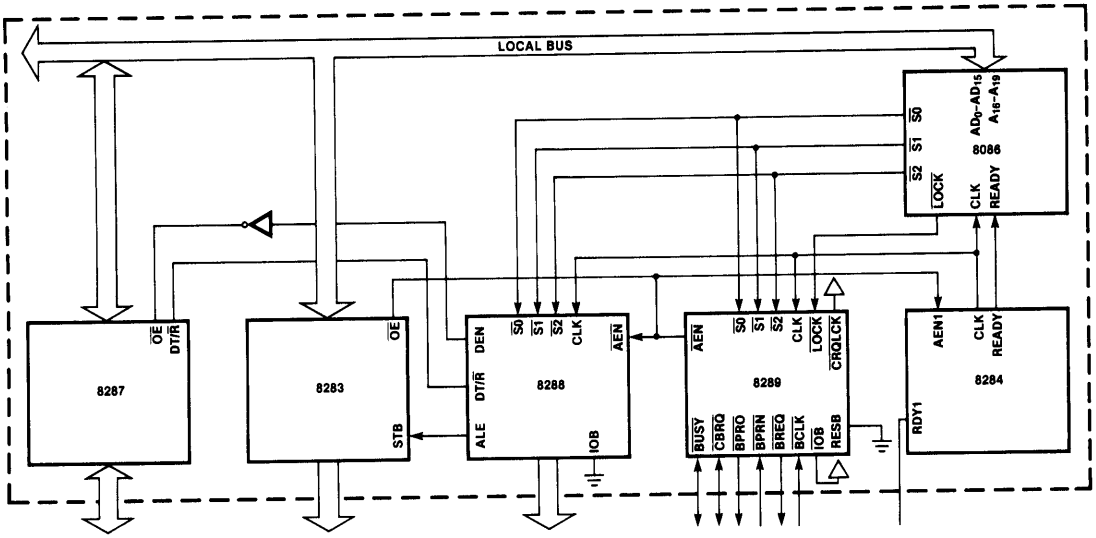


Figure 7. Single Multimaster Bus Interface

Figure 8 shows how a typical multi-processing system might be configured with the 8289 in the Single Bus mode. In the system there are three bus masters, each having the assigned priority as indicated—priority 1 being the highest and priority 3 being the lowest. Priority is established using the parallel priority scheme (ignore the dotted signal interconnect for the moment). Each bus arbiter monitors its associated processor and issues a bus request ($\overline{BREQ}$) whenever its processor wants the bus. A common clocking signal ($\overline{BCLK}$) runs to each of the arbiters in the system. It is from the falling edge of this clock that all bus requests are issued. Since all bus requests are made on the same clock edge, a valid priority can be established by the priority resolving circuitry by the next falling $\overline{BCLK}$ edge. Note that all multi-master system bus (MULTIBUS) input signals are considered to be valid at the falling edge of $\overline{BCLK}$. And that all multi-master system bus output signals are issued from the falling edge of $\overline{BCLK}$. With the parallel resolving module, arbiters 2 and 3 would issue their respective $\overline{BREQ}$s (Figure 9) on the falling edge of $\overline{BCLK\ 1}$, as shown. The outputs ($\overline{BPRN\ 1}$, $\overline{BPRN\ 2}$, and $\overline{BPRN\ 3}$) of the priority encoder-decoder arrangement change to reflect their new input conditions and need to be valid early enough in front of $\overline{BCLK\ 2}$ to guarantee the arbiter's setup time requirements. Since arbiter 2 at the time is the highest priority arbiter requesting the bus, bus priority is given to arbiter 2 ($\overline{BPRN\ 2}$ goes low), and since the bus was not busy ($\overline{BUSY}$ is high) at the time priority was granted to arbiter 2, arbiter 2 pulls $\overline{BUSY}$ inactive on $\overline{BCLK\ 2}$, thereby seizing the bus and excluding all other arbiters access to the bus. Once the bus is seized, arbiter 2 activates its $\overline{AEN}$. $\overline{AEN}$ going low directly enables the 8283 address latches and wakes up the 8288 Bus Controller. The bus controller enables the 8287 transceivers, waits until the address to command setup time has been established, and then enables its command drivers onto the bus.

If the serial priority resolving mode was used instead, much of the events that happened for the parallel priority resolving mode would be the same except, of course, there would be no parallel priority resolving module. Instead, the system would be connected as indicated in Figure 8 by the dotted signal lines connecting the $\overline{BPRO}$ of one arbiter to $\overline{BPRN}$ of the next lower priority arbiter.

The $\overline{BREQ}$ lines would be disconnected and the priority encoder-decoder arrangement removed. This arrangement is simpler than the parallel priority arrangement except that the daisy-chain propagation delay of the highest priority bus arbiter's $\overline{BPRO}$ to the lowest priority bus arbiter's $\overline{BPRN}$, including setup time requirement ($\overline{BPRN}$ to $\overline{BLCK}$), cannot exceed the $\overline{BCLK}$ period. In short, this means there are only so many arbiters that can be daisy-chained for a given $\overline{BCLK}$ frequency. Of course, the lower the $\overline{BCLK}$ frequency, the more arbiters can be daisy-chained. The maximum $\overline{BCLK}$ frequency is specified at 10 MHz, which would allow for three 8289 arbiters to be daisy-chained. In general, the number of arbiters that can be connected in the serial daisy-chain configuration can be determined from the following equation:

$$\overline{BCLK}\ \text{period} \geq \text{TBLPOH} + \text{TPNPO}\,(N-1) + \text{TPNBL}$$

where N = # of arbiters in system

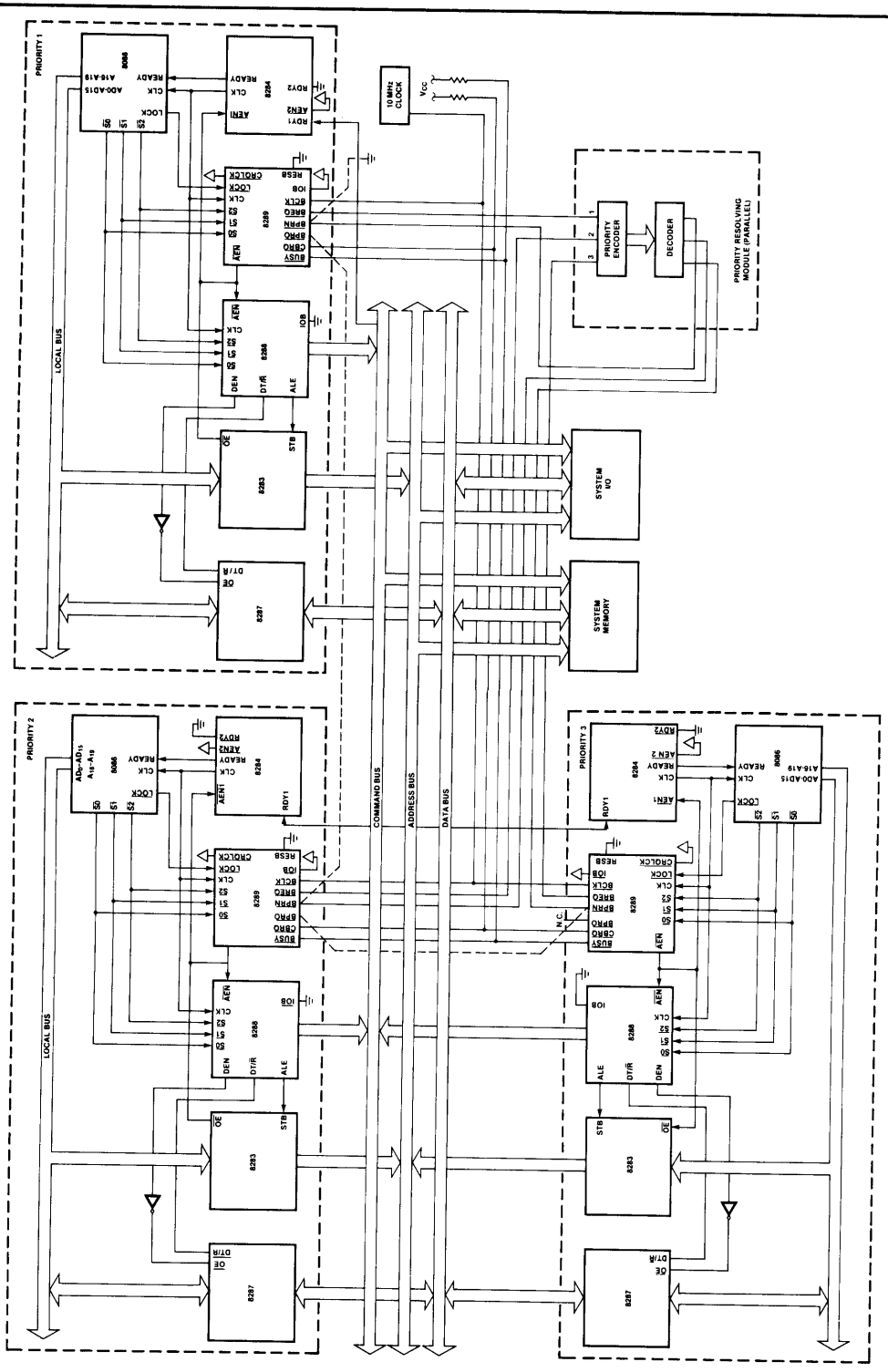


Figure 8. Multiprocessing System With 8289 In Single Bus Mode

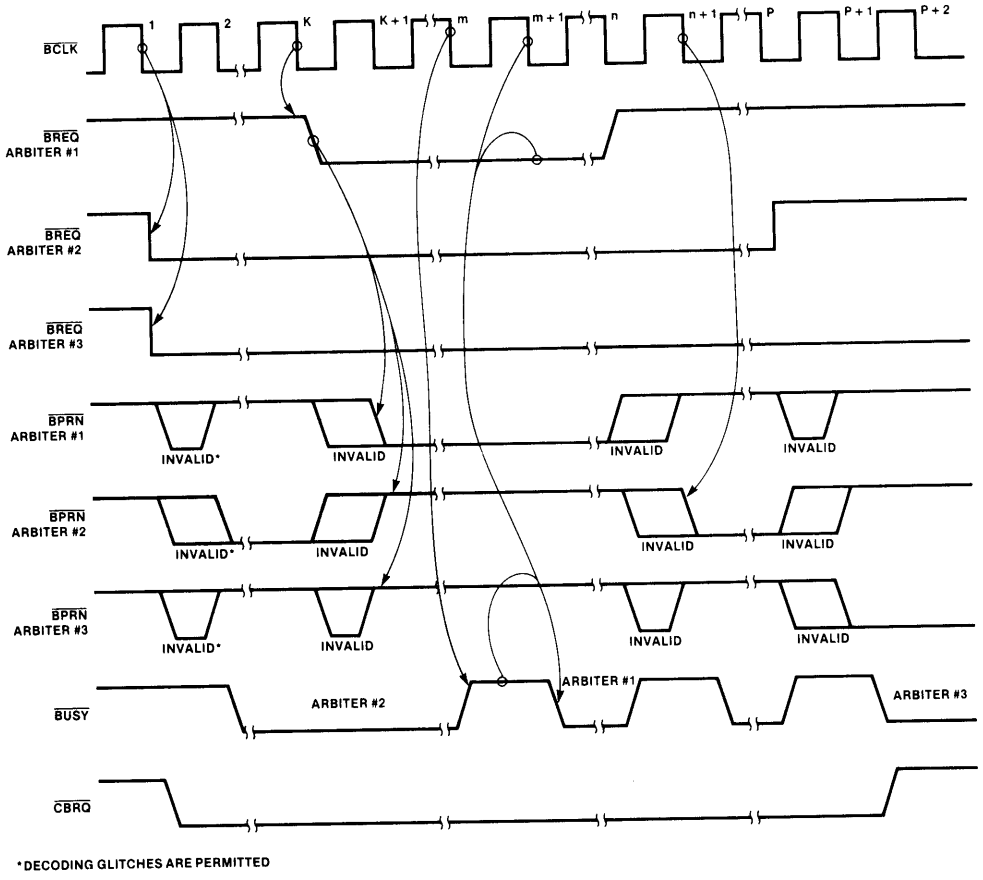


Figure 9. Example Timing For Figure 8

Returning to Figure 9, it can be seen that K $\overline{\text{BCLK}}$s later, arbiter 1 has decided to request the bus and its $\overline{\text{BREQ}}$, $\overline{\text{BREQ 1}}$, has gone low. Since arbiter 1 is of higher priority than arbiter 2, which presently has the bus, bus priority is reassigned by the priority module (or the daisy-chain approach in the serial priority) to arbiter 1. $\overline{\text{BPRN 1}}$ goes low and $\overline{\text{BPRN 2}}$ now goes high ($\overline{\text{BPRN 3}}$ remains high, even though decoding can cause it to glitch momentarily). The loss of priority instructs arbiter 2 that a higher priority arbiter wants the bus and that it is to release the bus as soon as its present transfer cycle is done. Since arbiter 2 cannot immediately release the bus, arbiter 1 must wait. In the particular case illustrated in Figure 9, arbiter 2 releases the bus (allows $\overline{\text{BUSY}}$ to go high) on clock edge M, and on clock edge M + 1, arbiter 1 now seizes the bus, pulling $\overline{\text{BUSY}}$ low. Arbiter 1 is the highest priority arbiter in the system and it now has the bus. Arbiters 2 and 3 still want the bus (their $\overline{\text{BREQ}}$s are both low).

How quickly arbiter 1 can acquire the bus is dependent upon the configuration and strapping options of the arbiter it is trying to acquire it from. For example, if the $\overline{\text{LOCK}}$ input to arbiter 2 was active (low) at the time, then arbiter 1, even though it was of higher priority, would not have acquired the bus until after $\overline{\text{LOCK}}$ was released (goes high). Effectively, $\overline{\text{LOCK}}$ locks the arbiter onto the bus once the bus has been acquired. $\overline{\text{LOCK}}$ will not force another arbiter to release the bus any sooner, it just prevents the bus from being given away no matter what the priority of the other arbiter. Another factor to be considered is where in the transfer cycle is the processor when the arbiter is instructed to give up the bus. Obviously, if the cycle had just started, it will take longer for the bus to be released than if the cycle was just ending. Another factor to be included in this consideration is the phase relationship of the processor's clock (CLK) to the bus clock ($\overline{\text{BCLK}}$). This relationship is examined in more detail later on. Table 1 lists the time

requirements for various arbiter actions such as bus acquisition and bus release (under $\overline{LOCK}$ and other circumstances) taking into account the phase relationships between CLK and $\overline{BCLK}$.

Bus Request (BREQ↓)	Mode	Delay (Max)	Delay (Min)
Status–BREQ↓	Single	2 $\overline{BCLK}$s	1 $\overline{BCLK}$
Status–BREQ↓	IOB	2 $\overline{BCLK}$s + ~1 CLK*	1 $\overline{BCLK}$ + ~½ CLK*
Status–BREQ↓	RESB	2 $\overline{BCLK}$s + ~2 CLKs†	1 $\overline{BCLK}$ + ~1½ CLKs†
Status–BREQ↓	IOB•RESB	2 $\overline{BCLK}$s + ~2 CLKs†	1 $\overline{BCLK}$ + 1½ CLKs†

*Request originates off of ϕ2 of T1 and $\overline{BREQ}$↓ occurs 1 $\overline{BCLK}$ (min) to 2 $\overline{BCLK}$s (max) thereafter. Depending upon where status occurs with respect to clock determines how long a time exists between status and ϕ2 of T1, and is anywhere from ½ CLK (min) to 1 CLK (max).

†Request originates off of T2•ϕ1 and $\overline{BREQ}$↓ occurs 1 $\overline{BCLK}$ (min) to 2 $\overline{BCLK}$s (max) thereafter. The same reasoning as used in the IOB mode is valid here.

Bus Release (BREQ↑)	Mode	Delay (Max)	Delay (Min)
Higher Priority (BPRN↑)	All	2 CLKs + 2 $\overline{BCLK}$s	1 CLK + 1 $\overline{BCLK}$
Lower Priority (CBRQ↓)	All	2 CLKs + 2 $\overline{BCLK}$s	1 CLK + 1 $\overline{BCLK}$
Surrender occurs once the proper surrender conditions exist.			

Table 1. Surrender and Request Time Delays

One signal which has been basically ignored to this point is $\overline{CBRQ}$. $\overline{CBRQ}$, like $\overline{BUSY}$, is an open-collector signal from the arbiter which is tied to the $\overline{CBRQ}$ signals of the other arbiters and to a pull-up resistor (see Figure 8). $\overline{CBRQ}$ is both an input and an output. As an output, $\overline{CBRQ}$ serves to instruct the arbiter presently on the bus that another arbiter wishes to acquire the bus. As an input, $\overline{CBRQ}$ serves to instruct the arbiter presently on the bus that another arbiter wants the bus. $\overline{CBRQ}$ is an input or output, dependent on whether the arbiter is on the bus or not (respectively), and is issued as a function of $\overline{BREQ}$. Thus, a lower priority arbiter requesting the bus already controlled by a higher priority arbiter will pull $\overline{CBRQ}$ low, as well as $\overline{BREQ}$. Even a higher priority arbiter will pull $\overline{CBRQ}$ low until it acquires the bus. Note, however, that the higher priority arbiter will acquire the bus through the reassignment of priorities — it being given priority and the other arbiter presently on the bus losing it. In effect, $\overline{CBRQ}$ serves to notify the arbiter that an arbiter of lower priority wants the bus.

If the arbiter presently on the bus is configured to react to $\overline{CBRQ}$ and the proper surrender conditions exist, the bus is released. When releasing the bus, the arbiter also turns off its $\overline{BREQ}$ ($\overline{BREQ}$ goes high) in order to allow priority to be established to the next lower arbiter requesting the bus. Such is the case shown in Figure 9. Whereas it was assumed that the proper surrender conditions did not exist for arbiter 2 when it had the bus, it is assumed that the proper conditions do exist during the time that arbiter 1 has the bus. Arbiter 2 had to give up the bus because an arbiter of higher priority was requesting it. Arbiter 1 surrenders the bus because the proper surrender conditions exist and a lower priority arbiter requested the bus by pulling $\overline{CBRQ}$ low. This is an assumed condition which is not otherwise shown in Figure 9. This is not an unrealistic condition. Normally, a higher priority arbiter will acquire the bus through the reassignment of priorities, while lower priority arbiters acquire the bus through $\overline{CBRQ}$.

Digressing for a moment, the 8289 Bus Arbiter will not voluntarily surrender the bus (except when the processor halts execution). As a result, it has to be forced off the bus. The 8289 Bus Arbiter does not generate a $\overline{BREQ}$ for each cycle. It generates a $\overline{BREQ}$ once and then hangs onto the bus. To do otherwise would require that $\overline{BREQ}$ be dropped (go high) after each transfer cycle so that if it did need to do another transfer cycle, another arbiter would automatically be assigned priority. This approach, however, entails certain overhead. Command to address setup and hold time must be prefixed and appended to each transfer cycle. Each transfer cycle would be characterized by first acquiring the bus, then establishing the setup time requirements, finally performing the transfer cycle, establishing the hold time requirements, and then releasing the bus (see Figure 10). If another transfer cycle was to immediately follow and if the arbiter still had priority, then the whole above procedure would be repeated. The end result would be wasted time as hold times following setup times (see Figure 10A). The approach taken by the 8289 Bus Arbiter of having to be forced off the bus, even when it is not using the bus (i.e., forced off by a lower priority arbiter), provides for greater bus efficiency. A lower priority arbiter having to force off another arbiter that is not using the bus but just hanging on to it, may not seem very efficient. In actuality it is a good trade-off. In many multimaster systems some bus masters occasionally demand the bus, while others demand the bus constantly. The bus master which constantly demands the bus may momentarily need not to access the bus. Why should that arbiter surrender the bus when chances are that the other bus masters which occasionally access the bus don't want it at the time? If it doesn't give up the bus, then it can momentarily cease access to the bus and then continue, without any performance penalty of having to reestablish control of the bus. The greater bus efficiency that it affords is well worth the added complexity (Figure 10B).

Returning to Figure 9, the combination of the proper surrender conditions existing and $\overline{CBRQ}$ being low, forced the higher priority arbiter, arbiter 1, off the bus. Arbiter 2, being of next higher priority and wanting the bus, acquired the bus on clock edge N + 1. If arbiter 1 decides to re-access the bus, it would reacquire the bus through the reassignment of priorities. This is not the case shown in Figure 9. Arbiter 1 has decided that it does not need the bus and does not renew its $\overline{BREQ}$. Arbiter 2, having acquired the bus through $\overline{CBRQ}$, is now the highest priority arbiter requesting the bus. As can be seen it is not the only arbiter requesting the bus. Arbiter 3 is still patiently waiting for the bus and $\overline{CBRQ}$ remains low. The same conditions that forced arbiter 1 off the

bus for arbiter 2 now forces arbiter 2 off the bus for arbiter 3. When the proper surrender conditions exist, arbiter 2 releases its $\overline{BREQ}$ and surrenders the bus to arbiter 3. Arbiter 3 acquires the bus on clock edge P + 1 and releases its $\overline{CBRQ}$. Since no other arbiter wants the bus (i.e., there is no other arbiter holding $\overline{CBRQ}$ low), $\overline{CBRQ}$ goes high (inactive). This would have also been true when arbiter 2 acquired the bus and released its $\overline{CBRQ}$ if arbiter 3 didn't want the bus.

In the Single interface, the arbiter monitors the processor's status lines, which are activated whenever the processor performs a transfer cycle. The arbiter, on detecting the status lines going active, will issue a $\overline{BREQ}$ if the status is not the HALT status. If the processor issues the HALT status, the arbiter will not request the bus, and if it has the bus, will release it.

This effectively concludes how arbiters interact to one another on the bus. Having examined the processor-to-arbiter interface, and arbiter-to-MULTIBUS (arbiter-to-arbiter) interaction, one interface is left, the internal interface of processor-related signals to that of MULTIBUS-related signals.

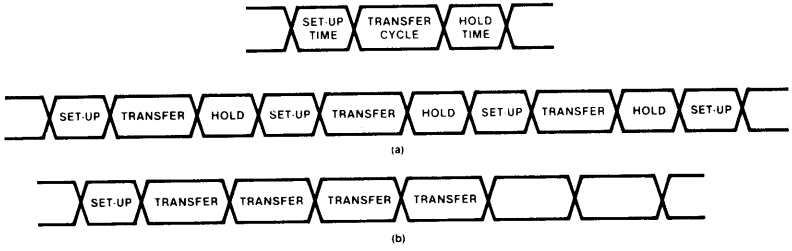
An important point to remember is that the processor has its own clock (CLK) and the multi-master system bus has its own ($\overline{BCLK}$). These two clocks are usually out of phase and of different frequencies. Thus, the arbiter must synchronize events occurring on one interface to events occurring on another interface. As a result of this back and forth synchronization, ambiguity can arise as to when events actually do take place.

Very simply, the 8289 arbiter operation can be represented as two events, requesting and surrendering. Figure 11 is a representation of the timing relationships involved. The request input is a function of the processor's clock and the surrender input is a function of either the bus clock or the processor's clock. To request the bus, the processor activates its status lines which in turn enables the request input. Depending upon the phase relationship between the occurrence of status (request active) and $\overline{BCLK}$, $\overline{BREQ}$ appears one to two $\overline{BCLK}$s later. As shown in Figure 12, the phase relationship between request and $\overline{BCLK}$ is such that the BRQ1 flip-flop may or may not catch request on the first $\overline{BCLK}$.*

If BRQ1 flip-flop does catch the request, then one $\overline{BCLK}$ later, $\overline{BREQ}$ goes low and one $\overline{BCLK}$ after that, $\overline{BUSY}$ goes low (it is assumed that priority is immediately granted and that the bus is available). If BRQ1 flip-flop does not catch the request, then request is caught on the next $\overline{BCLK}$ and $\overline{BREQ}$ goes low one $\overline{BCLK}$ later, followed by $\overline{BUSY}$ which also goes low one $\overline{BCLK}$ later. Note that $\overline{BREQ}$ and $\overline{BUSY}$ track, as $\overline{BREQ}$ is an input term for $\overline{BUSY}$. During bus acquisition, the surrender flip-flop is false (SURNDR Q = low) and $\overline{AEN}$ follows $\overline{BUSY}$.

Once the bus is acquired, the surrender circuitry is enabled so that when a valid surrender condition exists, the bus can be surrendered. The surrender circuitry synchronizes the surrender request to the processor's clock and drives SURNDR low. Like the acquisition circuitry, it takes from one to two processor clocks to generate SURNDR and depends upon the phase relationship between the surrender request and the processor's clock.

*The two bus request flip-flops, BRQ1 and BRQ2, are edge-triggered, high resolution flip-flops and serve to reduce the probability of walkout down to an acceptable level. Walkout occurs because $\overline{BCLK}$ is asynchronous with respect to request. If walkout does occur on BRQ1 flip-flop, the probability is high that the BRQ1 flip-flop will resolve itself prior to BRQ2 flip-flop being triggered. Even if BRQ1 flip-flop did not quite resolve itself, the probability of BRQ2 flip-flop walking out to an unacceptable point in time is itself low.



a) BUS UTILIZATION AS A RESULT OF HAVING TO REQUEST AND RELEASE THE BUS FOR EACH TRANSFER CYCLE. THIS PERMITS LOWER PRIORITY ARBITERS EASY ACCESS TO THE BUS SHOULD THE HIGHER PRIORITY ARBITER NO LONGER NEED THE BUS. HOWEVER, BUS EFFICIENCY IS POOR DUE TO THE ARBITER THRASING ON AND OFF OF THE BUS FOR EACH TRANSFER CYCLE.

b) 8289 BUS UTILIZATION IS MORE EFFICIENT IN THAT THE ARBITER HAS ONLY TO ACQUIRE THE BUS ONCE. THE 8289 HANGS ONTO THE BUS UNTIL FORCED OFF. THIS APPROACH ADDS A LITTLE MORE COMPLEXITY TO THE SYSTEM INASMUCH AS SOME MEANS MUST BE PROVIDED FOR LOWER PRIORITY ARBITERS TO FORCE THE HIGHER PRIORITY ARBITER OFF OF THE BUS WHEN IT IS NOT USING IT. THE ADDED COMPLEXITY IS WELL WORTH THE BUS EFFICIENCY AND SYSTEM FLEXIBILITY IT AFFORDS. THE 8289 ARBITER CAN BE CONFIGURED TO HAVE THE TRANSFER TIMING AS SHOWN IN (a) (IMITATING THE METHOD 8218 AND 8219 USES, BUS ARBITERS FOR 8080 AND 8085 RESPECTIVELY) BY STRAPPING ANYRQST HIGH AND $\overline{CBREQ}$ LOW.

Figure 10. Two Techniques For Doing Multibus Transfer Cycles

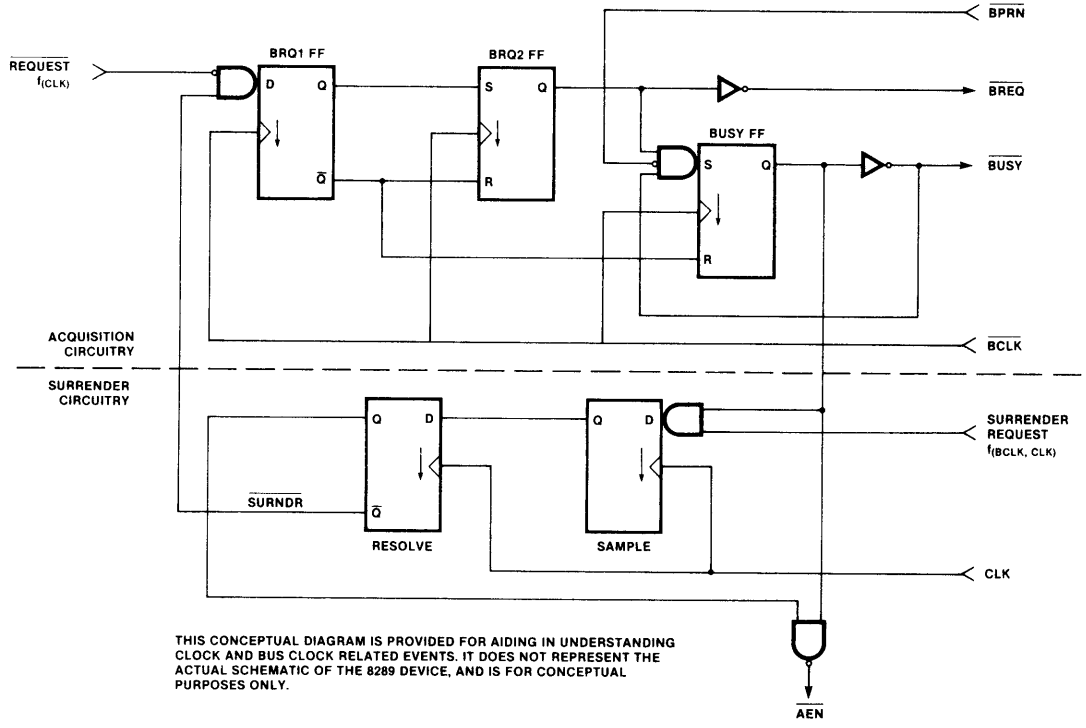
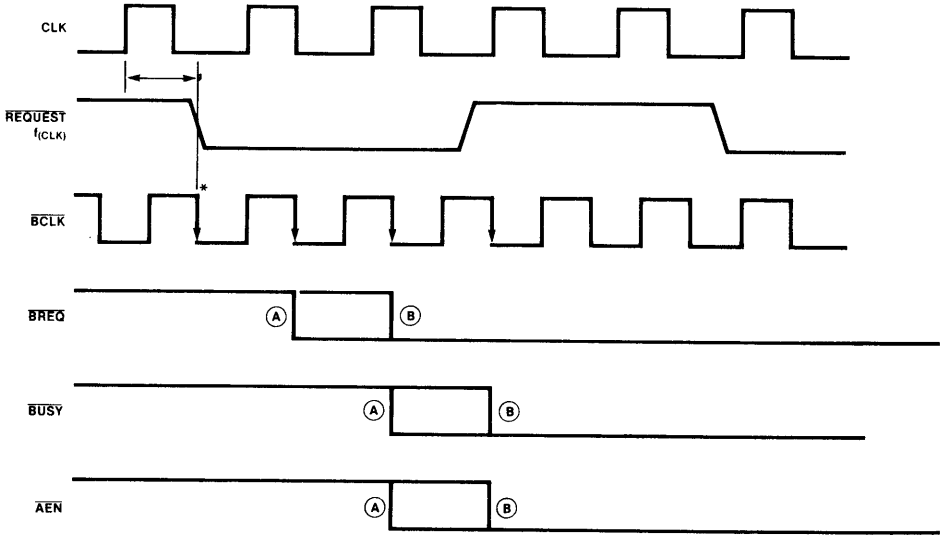


Figure 11. Symbolic Representation of Internal 8289 Timing



*WHEN THE REQUEST OCCURS SIMULTANEOUSLY WITH BCLK, BCLK MAY OR MAY NOT CATCH THE REQUEST. IF IT DOES, THE WAVEFORMS FOLLOW THOSE SHOWN DESIGNATED BY (A). IF NOT, THE REQUEST IS PICKED UP ON THE NEXT EDGE OF BCLK AND THE WAVEFORMS FOLLOW THOSE SHOWN DESIGNATED BY (B).

Figure 12. Results Of An Asynchronous Event

Having synchronized the surrender request to the processor's clock to generate SURNDR, SURNDR is then synchronized to $\overline{BCLK}$ to reset the BUSY and BRQ flip-flops. When BUSY-Q goes low, the surrender circuitry is reset which in turn re-enables the request input. The timing in Figure 13 shows the surrender request input going high on the falling edge of the clock. If the Sample flip-flop was able to catch the surrender request on the edge of clock 1, then SURNDR would be generated (go low) on clock edge 2. If not, SURNDR would be generated on clock edge 3. SURNDR going low on clock edge 2 will be, for ease of discussion, referred to as SURNDR a and SURNDR going low on clock edge 3 will be referred to as SURNDR b. As can be seen from Figure 13, SURNDR a just happens to go low on $\overline{BCLK}$ edge 2. Since SURNDR is used to reset the BRQ flip-flops, which are clocked by the falling edge of $\overline{BCLK}$, the BRQ1 flip-flop may or may not catch SURNDR a on $\overline{BCLK}$ edge 2. If it does, then BRQ and $\overline{BUSY}$ go high on BCLK edge 3 which, for convenience, will be called $\overline{BREQ}$ a or $\overline{BUSY}$ a. If not, then $\overline{BREQ}$ and $\overline{BUSY}$ will go high on $\overline{BCLK}$ edge 4, which will be referred to as $\overline{BREQ}$ b or $\overline{BUSY}$ b, respectively. SURNDR b occurs early enough to assure that $\overline{BUSY}$ and $\overline{BREQ}$ are reset on $\overline{BCLK}$ edge 5, which will be referred to as $\overline{BUSY}$ b1 and $\overline{BREQ}$ b1. Depending upon when $\overline{BUSY}$ goes high, determines when the surrender circuitry is reset and how soon the next $\overline{BREQ}$ can be generated. $\overline{BUSY}$ a1 causes SURNDR c to occur where shown and SURNDR c in turn would allow the earliest bus request to occur at $\overline{BREQ}$ c1. At the other extreme, $\overline{BUSY}$ b1 allows the earliest bus request to occur at $\overline{BREQ}$ e1.

Table 1 summarizes the maximum and minimum delays for bus request, once the proper request and surrender conditions exist. Table 2 lists the proper surrender conditions.

Mode	Surrender Conditions
Single	HALT state, loss of BPRN, TI•CBREQ
IOB	HALT state, loss of BPRN, TI•CBREQ, I/O Command•CBRQ
RESB	HALT state, loss of BPRN, TI•CBREQ, (SYSB/RESB = 0)•CBRQ
IOB•RESB	HALT state, loss of BPRN, TI•CBREQ, (SYSB/RESB = 0)•CBREQ, I/O Command•CBRQ

Table 2. Surrender Conditions

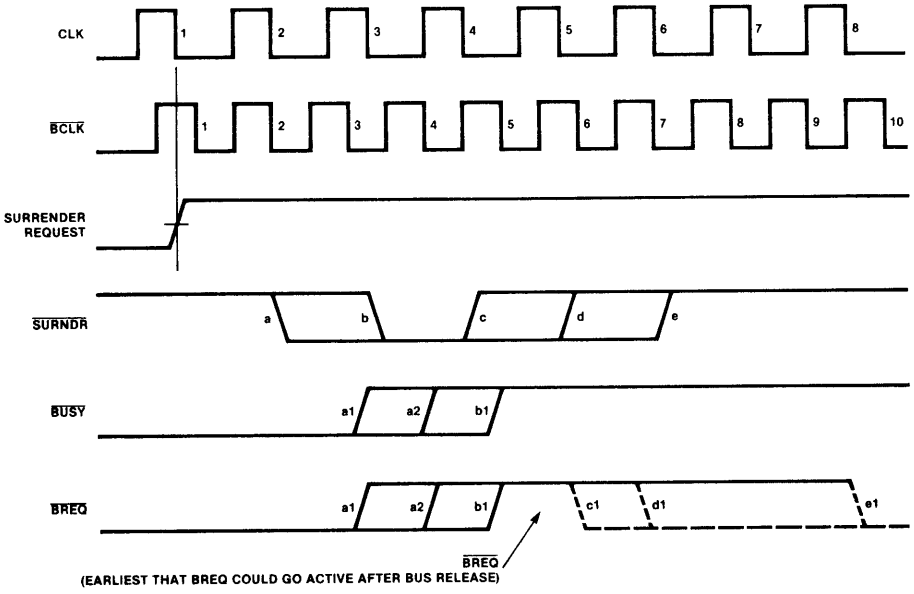


Figure 13. Asynchronous Bus Release

IOB INTERFACE

Now that the processor-arbiter, arbiter-system bus and internal arbiter timings have been discussed, it is appropriate to consider the other interfaces that the 8289 Bus Arbiter provides.

In the IOB mode, the processor communicates and controls a host of peripherals over the peripheral bus. When the I/O processor needs to communicate with system memory, it is done so over the system memory bus. Figure 14 shows a possible I/O processor system configuration, utilizing the 8089 I/O processor in its REMOTE mode. Resident memory exists on the peripheral bus in order that canned I/O routines and buffer storage can be provided. Resident memory is treated as an I/O peripheral. When a peripheral device needs servicing, the I/O processor accesses resident memory for the proper I/O driver routine and services the device, transmitting or storing peripheral data in buffer storage area of resident memory. The resident memory's buffer storage area could then be emptied or replenished from system memory via the system bus. Using the IOB interface allows an I/O processor the capability of executing from local memory (on the peripheral bus) concurrently with the host processor.

Timing in this mode is no different from timing in the SINGLE BUS mode. The only difference lies in the request and surrender conditions. The arbiter extends the single bus mode conditions to qualify when the system bus is requested and adds on additional surrender conditions. The system bus is only requested during system bus commands (the arbiter decodes the processor's status lines) and, in addition to the other surrender terms, the arbiter permits surrender to occur during I/O bus (or local bus) commands, when the I/O processor is using its own local bus.

Like the arbiter, the bus controller must also be informed of the mode it is operating in. In the IOB mode, the 8288 bus controller issues I/O bus commands independently of the state of $\overline{AEN}$ from the arbiter. It is assumed that all I/O bus commands are intended for the I/O bus and hence there is a separate I/O command bus from the controller. All I/O bus commands are sent directly to the I/O bus and are not influenced by $\overline{AEN}$. System bus commands are assumed as going to the system bus. Since system bus commands are directed to the system bus, they must still be influenced by $\overline{AEN}$ and the arbitration mechanism provided by the 8289.

As an example, suppose the processor issues an I/O bus command. The 8288 Bus Controller generates the necessary control signal to latch the I/O address and configure the transceivers in the correct direction. In the IOB mode, the multiplexed MCE/$\overline{PDEN}$ pin of the 8288 becomes $\overline{PDEN}$ (peripheral data enable) and serves to enable the I/O bus's data transceivers during I/O bus commands. DEN similarly serves to enable the system bus's data transceivers during memory commands. $\overline{PDEN}$ and DEN are mutually exclusive, so it is not possible for both sets of transceivers to be on, thereby avoiding contention between the two sets. Since the I/O bus commands are generated independently of $\overline{AEN}$ In the IOB mode, the I/O bus has no delay effects due to the arbiter. During this time in which the processor is accessing memory the arbiter, if it already has the bus, will permit it to be surrendered to either a higher or lower priority independently of where the processor is in

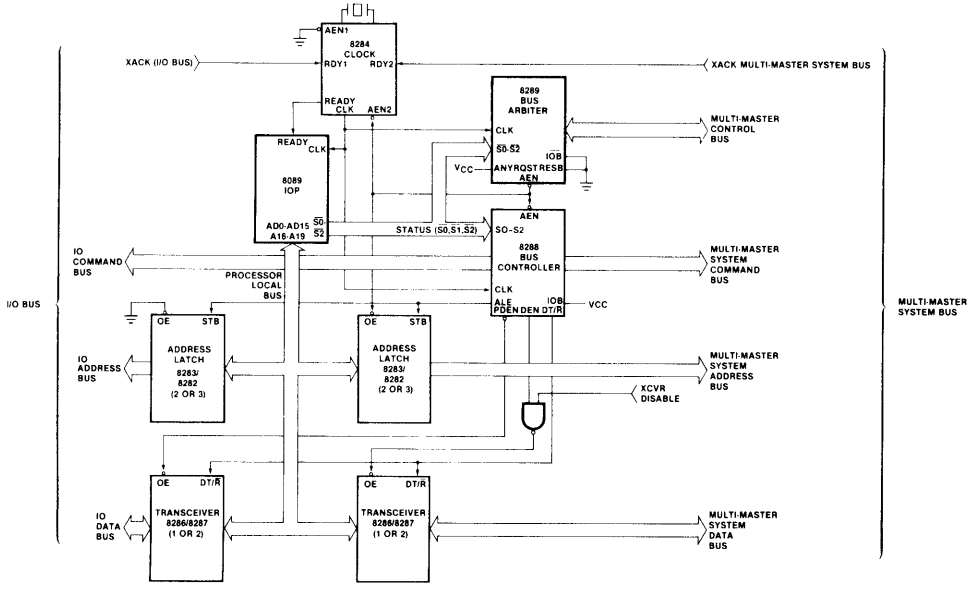


Figure 14. 8289 Configured In I/O Bus Mode With 8089 I/O Processor

its transfer cycle (i.e., independent of the machine state).* If the arbiter does not already have the bus, it will make no effort to acquire the bus.

If the processor issues a memory command instead, the same set of events take place, except that 1) the system bus's data transceivers are enabled instead of the peripherals bus's data transceivers, and 2) when the command is issued depends upon the state of the arbiter. In both cases of I/O bus commands and system bus commands, the address generated for that command is latched into both sets of address latches, the system bus's address latches, and the peripherals bus's address latches. For each command (regardless of command type), an address is put out on the I/O bus and on the system bus if the arbiter has the bus at that particular time. However, the bus controller only issues a command to one of the buses and hence, no ill effects are suffered by addressing both buses.

If the arbiter already has the system bus when a system bus command is issued, no delays due to the arbiter will be noticed by the processor. If the arbiter doesn't have the bus and must acquire it, then the processor will be delayed (via the system bus command being delayed by the bus controller through $\overline{AEN}$ from the arbiter) until the arbiter has acquired the bus. The arbiter will then permit the bus controller to issue the command and the transfer cycle continues.

RESB INTERFACE

The non-I/O processors in the 8086 family can communicate with both a resident bus and a multi-master system bus. Two bus controllers would be needed in such a configuration as shown in Figure 15. In such a system configuration the processor would have to access to memory and peripherals of both buses. Address mapping techniques can be applied to select which bus is to be accessed. The SYSB/$\overline{RESB}$ (system bus/resident bus) input on the arbiter serves to instruct the arbiter as to whether or not the system bus is to be accessed. It also enables or disables commands from one of the bus controllers.

In such a system configuration, it is possible to issue both memory and I/O commands to either bus and as a result, two bus controllers are needed, one for each bus. Since the controllers have to issue both memory and I/O commands to their respective buses, the IOB options on the controllers are strapped off (IOB is low). The arbiter, too, has to be informed of the system configuration in order to respond appropriately to system inputs and has its RESB option strapped on (RESB is high). The arbiter's IOB option is strapped inactive ($\overline{IOB}$ is high). Strapping the arbiter into the resident bus mode enables the arbiter to respond to the state of the SYSB/$\overline{RESB}$ input. Depending upon the state of this input, the arbiter either requests and acquires the system bus or permits the surrendering of that bus.

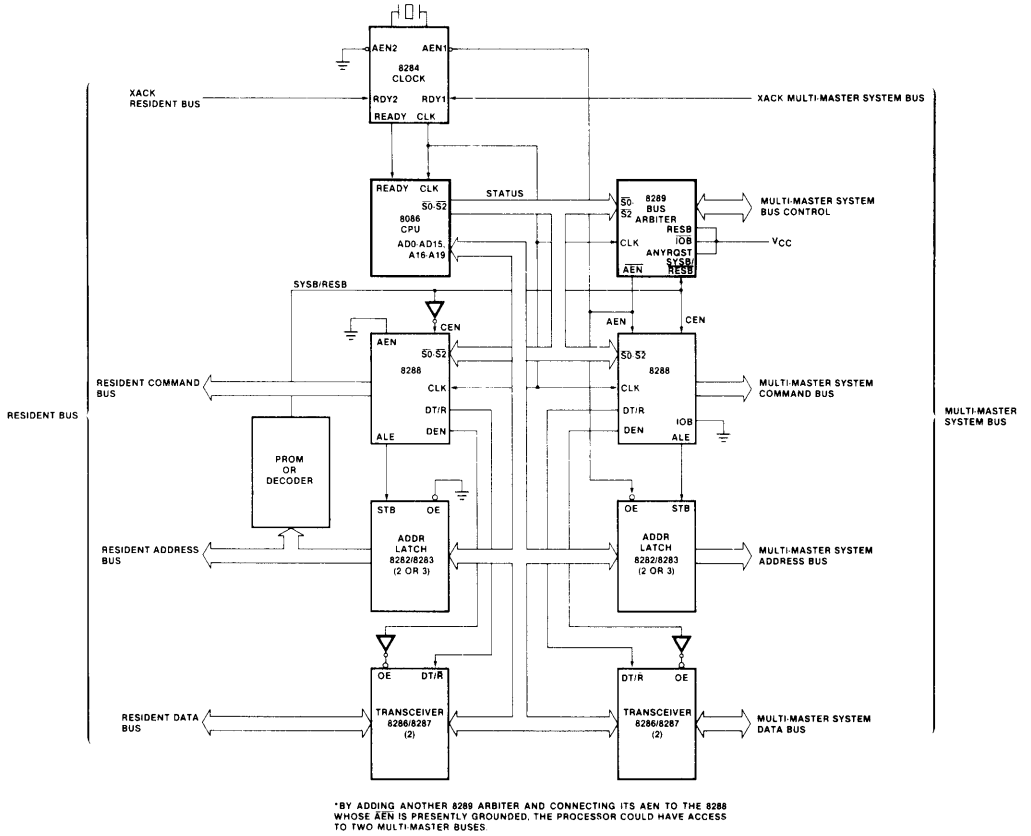
*Under other circumstances, bus surrendering would only be permitted during the period from where address to command hold time has been established just prior to where the next command would be issued.

In the system shown in Figure 15, memory mapping techniques are applied on the resident bus side of the system rather than on the multiprocessor or system bus side. As mentioned earlier in the IOB interface, both sets of address latches (the resident bus's address latches and the system bus's address latches) are latched with the same address; in this case, by their respective bus controllers.* The system bus's address latches, however, may or may not be enabled depending upon the state of the arbiter. The resident bus's address latches are always enabled, hence the address mapping technique is applied to the resident bus.

Address mapping techniques can range in complexity from a single bit of the address bus (usually the most significant bit of the address), to a decoder, to a PROM. The more elaborate mapping technique, such as PROM, provides segment mapping, system flexibility, and easy mapping modifications (simply make a new PROM).

In actual operation, both bus controllers respond to the processor's status lines and both will simultaneously issue an address latch strobe (ALE) to their respective address latches. Both bus controllers will issue command and control signals unless inhibited. The purpose of the address mapping circuitry is to inhibit one of the bus controllers before contention or erroneous commands can occur. The transceivers are enabled off the same clock edge the commands are issued, namely $\phi 1$ of T2 (Figure 16). The address is strobed into the address latches by ALE. ALE is activated as soon as the processor issues status, and is terminated on $\phi 2$ of of T1. From when ALE is issued, plus the propagation delay of the address latches, determines where the address is valid. The time from which the address is valid to where control and commands are issued determines how much settling time is available for the address mapping circuitry. The mapping circuitry must inhibit (via CEN) one of the bus controllers prior to where controls and commands are issued. Part of the settling time (see Figure 16) is consumed as a setup time requirement to the bus controllers. As it turns out, CEN (command enable) can be disqualified as late as on the falling edge of clock (the leading edge of $\phi 1$ of T2) without fear of the bus controller issuing any commands or transceiver control signals. In systems (8 MHz) where less time is available for the address mapping circuitry, the address latches can be bypassed, hooking the mapping circuitry straight onto the processor's multiplexed address/data bus (the local bus) and using ALE to strobe the mapping circuitry. This would avoid the propagation delay time of the transceivers. Besides needing to inhibit one of the bus controllers, the arbiter needs to be informed of the address mapping circuitry's decision. Depending upon that decision, the arbiter acquires or permits the release of the system bus.

*A simpler system with an 8086 or 8088 can exist, if it is desirable to only have PROM, ROM, or a read only peripheral interface on the resident bus. The 8086 and 8088 additionally generate a read signal in conjunction with the 8288 control signals. By using this read signal and memory mapping, the 8086 or 8088 could operate from local program store without having the contention of using the system bus.



*BY ADDING ANOTHER 8289 ARBITER AND CONNECTING ITS AEN TO THE 8288 WHOSE AEN IS PRESENTLY GROUNDED, THE PROCESSOR COULD HAVE ACCESS TO TWO MULTI-MASTER BUSES.

Figure 15. 8289 Configured In Resident Bus Mode

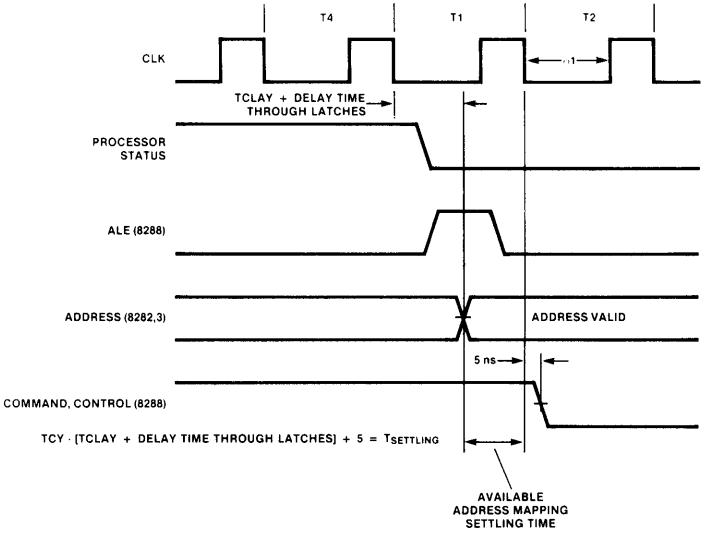


Figure 16. Time Available For Address Mapping Prom

The arbiter is informed of this decision via its SYSB/$\overline{\text{RESB}}$ input. If the memory mapping circuitry selects the resident bus, then SYSB/$\overline{\text{RESB}}$ input to the arbiter and CEN input of the system bus controller are brought low; and the CEN Input of the resident bus controller is brought high. The commands and control signals of the resident bus are now enabled and those of the system bus are disabled. In addition, with the arbiter being informed that the transfer cycle is occurring on the resident bus, the system bus is permitted to be surrendered. Glitching is permitted on the SYSB/$\overline{\text{RESB}}$ input of the arbiter up until ϕ1 of T2. Thereafter, only clean transitions can occur on the input.* So, if mapping circuitry can settle prior to ϕ1 of T2, there is no need to be concerned over glitching. If the mapping circuitry is unable to settle prior to this time, then the designer must guarantee a clean transition on the SYSB/$\overline{\text{RESB}}$ input.

INTERFACE TO TWO MULTI-MASTER BUSES

The interface of an 8086 family processor to two multisystem buses is simply an extension of the resident bus interface. The only difference is that now two arbiters are needed, one for each multi-master bus, and the address mapping circuitry must acquire its input straight off the processor's multiplexed address/data bus (the local bus), using ALE as an address strobe input. Figure 17 depicts how such a system might be configured.

Figure 17 illustrates the use of the 8289 in a system environment in three of its four modes. The host 8086 CPU (priority 3) is using the 8289 in its single bus multimaster mode, while an 8089 I/O processor is using the 8289 in its IOB mode. A work station based on an 8088 processor uses the 8289 in it system/resident bus mode. This diagram represents a hypothetical system wherein there can exist more than one work station (only one shown). Each work station shares system resources and I/O. The lowest priority processor (8086) would provide supervisory functions and system control, i.e., allow operator intervention into the system resources. A work station would call in assemblers and compilers or application programs as needed. When compiled or assembled, the results are transferred to the I/O station for output, thus freeing up a work station for another user.

*In certain memory mapping techniques, the CENs of the bus controllers are controlled differently from the SYSB/$\overline{\text{RESB}}$ input of the arbiter. In short, CEN is brought low automatically to both bus controllers, thereby disabling their command and control outputs. This permits a longer settling time for the memory mapping circuitry, since both controllers are disabled. When the mapping circuitry settles, sometime after ϕ1 of T2, one of the bus controllers and its associated bus arbiter (if one exists) is enabled. After ϕ1 of T2, the arbiter can only permit clean transitions on the SYSB/$\overline{\text{RESB}}$ input line.

If one work station is used, the serial priority resolving technique could be used between the 8289 Bus Arbiters (shown in dotted lines). If more than one work station is desired, it would be necessary to either slow down the system bus clock to accommodate the additional arbiters, or resort to the parallel resolving technique (as shown).

WHEN TO USE THE DIFFERENT MODES

Single Bus Multi-Master Interface

This mode is the simplest and is sufficient for systems where a multiprocessing environment exists and the system bus bandwidth is sufficient to handle the peak concurrent requirements of a multi-master environment. This solution can provide an inexpensive solution for multi-masters to access an expensive I/O device. If, however, the system bus bandwidth is exceeded, the IOB or system/resident modes should be considered.

IOB Mode

The IOB mode is ideal when the bus can be separated into an I/O bus and memory or system bus. This mode is commonly used with the 8089 I/O processor in its REMOTE configuration to separate the I/O space from memory space. With the 8089, all instructions operate on either system or I/O address space. 64K bytes of I/O space can be accessed by the processors in the 8086 family.

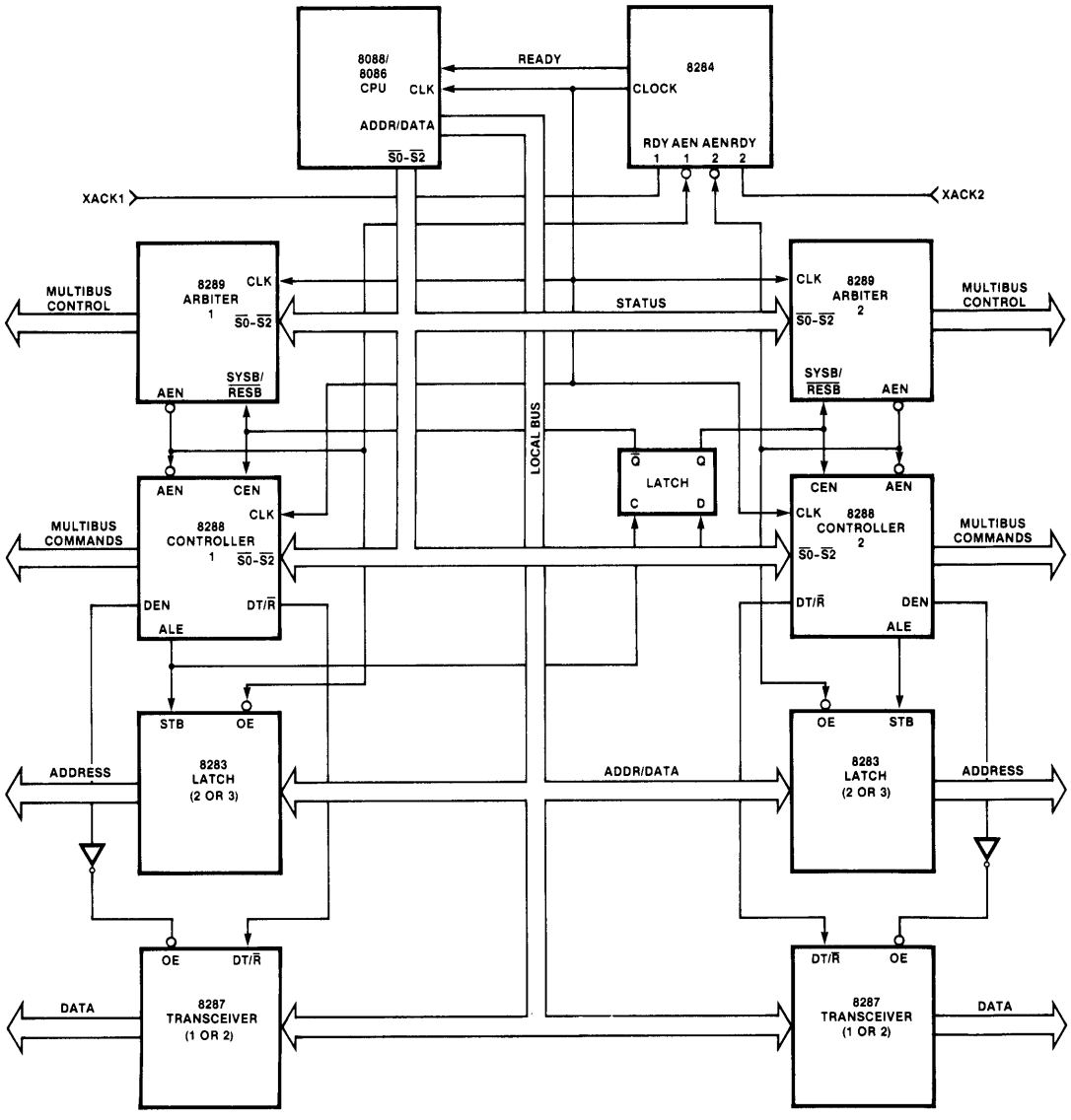
The remaining processors in the 8086 family are constrained to using only I/O instructions when referencing I/O space. If this is a limitation, and it is desirable to remove some of the processor functions to its private resources, the resident bus mode should be considered.

Resident Bus Mode

The resident bus mode allows for maximum flexibility for a CPU device, giving it both access to its own local resources with full instruction set capability, and the system resources. The CPU can work from its own local resources without contention on the system bus. By using a PROM for memory mapping, memory space can be easily altered in this mode. This mode requires the use of a second 8288 bus controller chip.

CONCLUSION

The 8289 brings a new dimension to microcomputer architecture by allowing the advanced 8/16-bit microprocessors to play easily in a multi-master, multiprocessing environment. With the flexible modes of the 8289, a user can define one of several bus architectures to meet his cost/performance needs. Modularity, improved system reliability and increased performance are just a few of the benefits that designing a multiprocessing system provides.



MEMORY MAPPING DECODING IS SHOWN TAKING PLACE DIRECTLY OFF OF THE PROCESSOR'S LOCAL MULTIPLEXED ADDRESS/DATA BUS.

Figure 17. Using 8289s To Interface To Two Multimaster System Buses.

THIS PAGE LEFT INTENTIONALLY BLANK

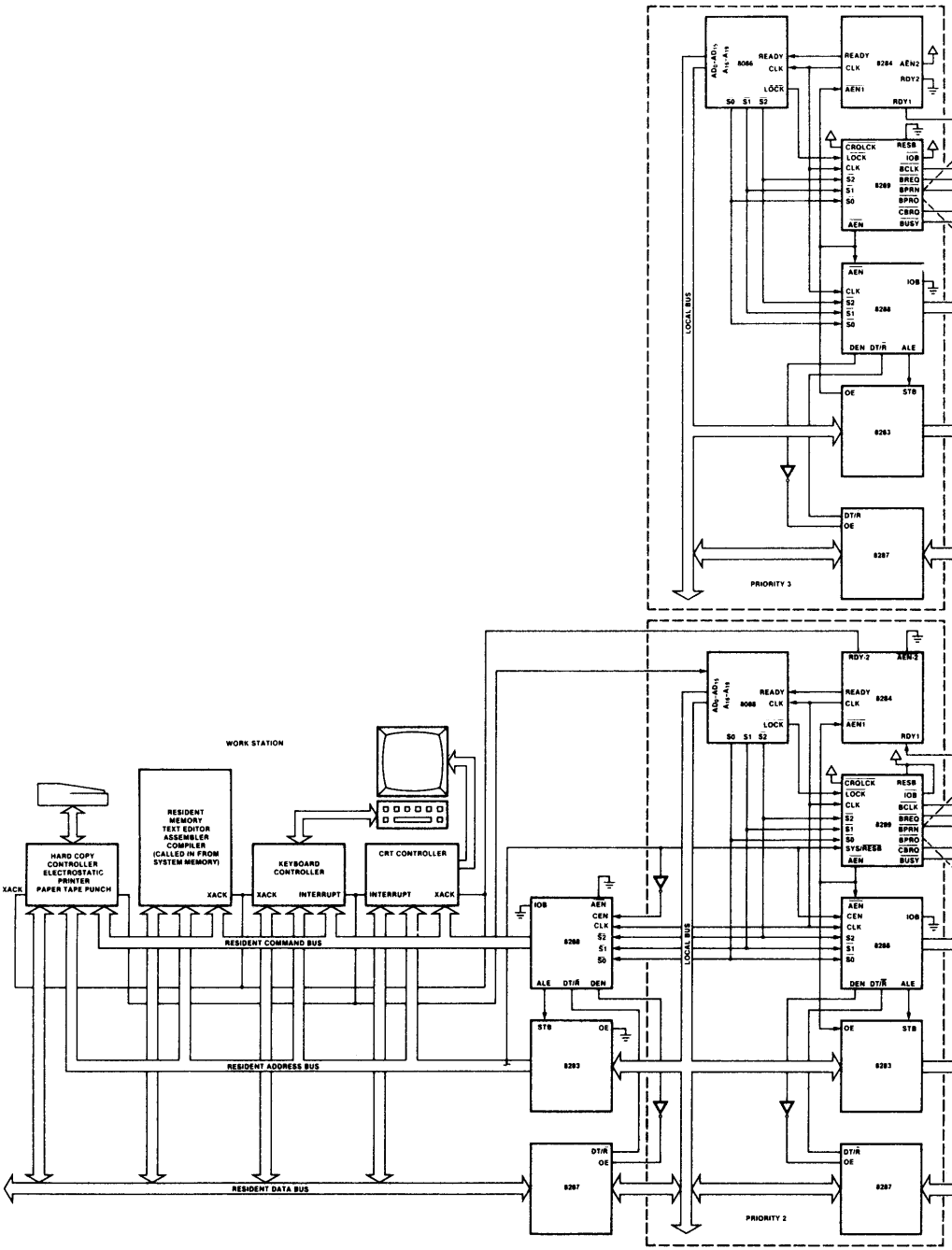


Figure 18. 8289 Used In Each Of 3 Modes, Single Bus, I/O Bus, and Resident Bus Modes Implementing A Hypothetical Multimaster Bus System

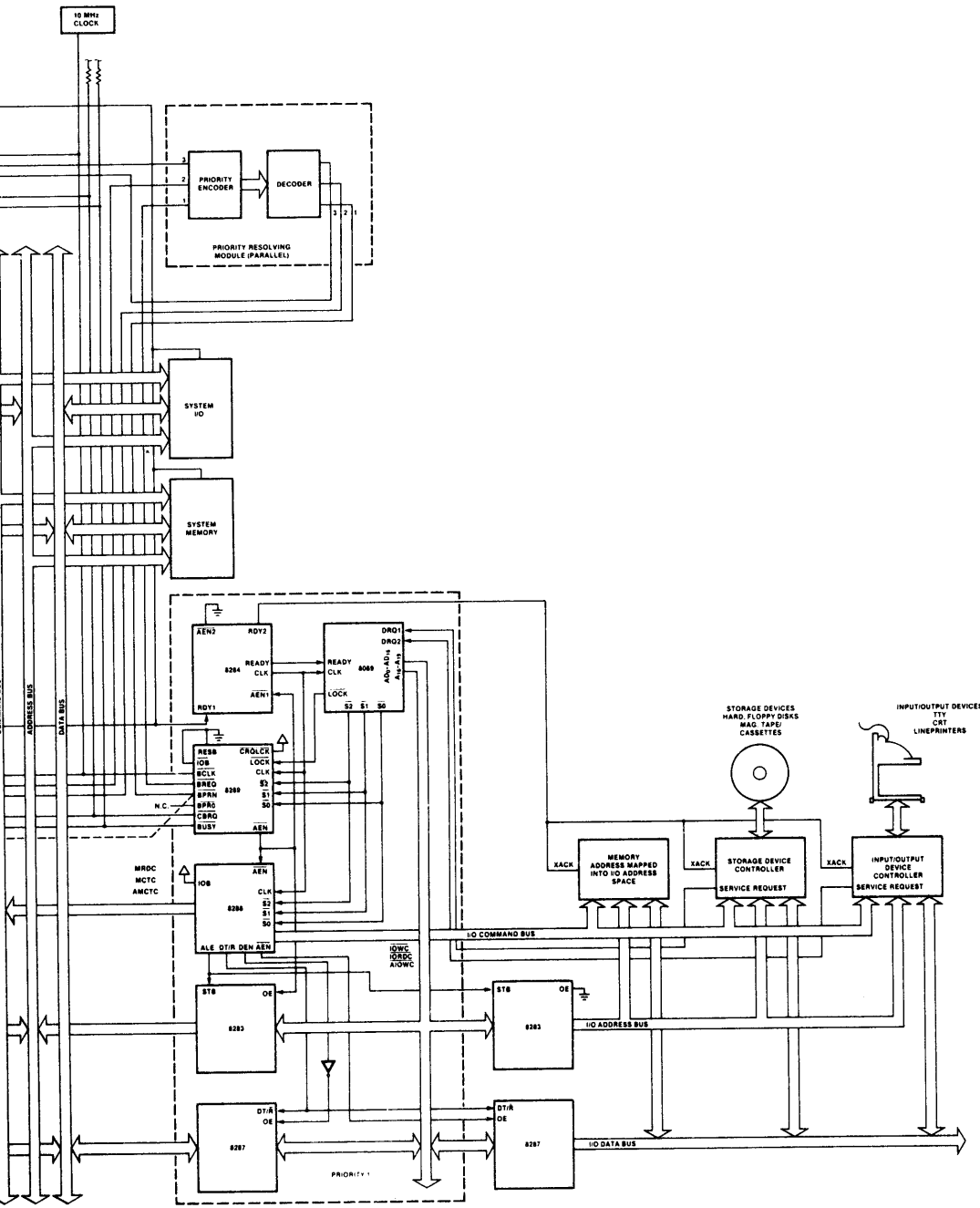


Figure 18. 8289 Used In Each Of 3 Modes, Single Bus, I/O Bus, and Resident Bus Modes Implementing A Hypothetical Multimaster Bus System